Data Structure

Lecture#21: Searching 2 (Chapter 9)

U Kang
Seoul National University

In This Lecture

- Concept and heuristics of self organizing list which speeds up searching
- Motivation and main idea of hashing
- Choosing good hash functions

Self-Organizing Lists

- Self-organizing lists modify the order of records within the list based on the actual pattern of record accesses.
  - Goal: minimize search time

- Self-organizing lists use a heuristic for deciding how to reorder the list.
  - Idea: move frequent items to front

Heuristics

1. Count: Order by actual historical frequency of access.
2. Move-to-Front: When a record is found, move it to the front of the list.
3. Transpose: When a record is found, swap it with the record ahead of it.

Heuristics - Example

- Assume we have eight records (A to H) which are initially in alphabetical order
- Consider the access pattern: $F\ D\ F\ G\ E\ G\ F\ A\ D\ F\ G\ E$

- Count heuristic?
- Move-to-front heuristic?
- Transpose heuristic?

Count Heuristic

- Consider the access pattern: $F\ D\ F\ G\ E\ G\ F\ A\ D\ F\ G\ E$

Cur Input	Order After Processing Cur Input	Cost
F	F(1) A(0) B(0) C(0) D(0) E(0) G(0) H(0)	6
D	F(1) D(1) A(0) B(0) C(0) E(0) G(0) H(0)	5
F	F(2) D(1) A(0) B(0) C(0) E(0) G(0) H(0)	1
G	F(2) D(1) G(1) A(0) B(0) C(0) E(0) H(0)	7
E	F(2) D(1) G(1) E(1) A(0) B(0) C(0) H(0)	7
G	F(2) G(2) D(1) E(1) A(0) B(0) C(0) H(0)	3
F	F(3) G(2) D(1) E(1) A(0) B(0) C(0) H(0)	1
A	F(3) G(2) D(1) E(1) A(1) B(0) C(0) H(0)	5
D	F(3) G(2) D(2) E(1) A(1) B(0) C(0) H(0)	3
F	F(4) G(2) D(2) E(1) A(1) B(0) C(0) H(0)	1
G	F(4) G(3) D(2) E(1) A(1) B(0) C(0) H(0)	2
E	F(4) G(3) D(2) E(2) A(1) B(0) C(0) H(0)	4

Total cost: 45 comparisons

Move-to-front Heuristic

- Consider the access pattern: *F D F G E G F A D F G E*

Cur Input	Order After Processing Cur Input	Cost
F	F A B C D E G H	6
D	D F A B C E G H	5
F	F D A B C E G H	2
G	G F D A B C E H	7
E	E G F D A B C H	7
G	G E F D A B C H	2
F	F G E D A B C H	3
A	A F G E D B C H	5
D	D A F G E B C H	5
F	F D A G E B C H	3
G	G F D A E B C H	4
E	E G F D A B C H	5

Total cost: 54 comparisons

Transpose Heuristic

- Consider the access pattern: $F\ D\ F\ G\ E\ G\ F\ A\ D\ F\ G\ E$

Cur Input	Order After Processing Cur Input	Cost
F	A B C D F E G H	6
D	A B D C F E G H	4
F	A B D F C E G H	5
G	A B D F C G E H	7
E	A B D F C E G H	7
G	A B D F C G E H	7
F	A B F D C G E H	4
A	A B F D C G E H	1
D	A B D F C G E H	4
F	A B F D C G E H	4
G	A B F D G C E H	6
E	A B F D G E C H	7

Total cost: 62 comparisons

Text Compression Example

- Application: Text Compression.
- Keep a table of words already seen, organized via Move-to-Front heuristic.
  - If a word not yet seen, send the word.
  - Otherwise, send (current) index in the table.
- The car on the left hit the car I left.
- The car on 3 left hit 3 5 I 5.
- This is similar in spirit to Ziv-Lempel coding.

Searching in Sets

- For dense sets (small range, high percentage of elements in set),
- Can use logical bit operators.
  - A AND B : A & B
  - A OR B : A | B
  - A – B : A &~B
- Example: To find primes (between 1 and 15) that are odd numbers, compute:
  - 0011010100010100 & 0101010101010101

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	1	0	1	0	0	0	1	0	1	0	0

Hashing

- Given a key *k*, can we search k in a constant time?
  - Yes!
  - We can do it by hashing. It is faster than binary search , QBS, and sequential search
- Hash table HT is an array that holds the records
  - HT has M slots (slots numbered from 0 to M-1)
- Hash function *h* maps key *K* to a number (position)
  - $0 \leq h(K) \leq M - 1$
  - E.g., h(*K*) = *K* % M
- Goal of a hashing system: arrange things such that for a given key *K*, and *i* = h(*K*), the record for the key *K* is located in HT[*i*]
  - Then, the searching time would be constant

Hashing

- Goal of a hashing system: arrange things such that for a given key K, and $i = h(K)$, the record for the key K is located in HT[i]
- Collision: two different keys k_1 and k_2 map to a slot
  - $h(k_1) = \beta = h(k_2)$
- Finding a record with key value K by hashing:
  - Compute the table location $h(K)$
  - Starting with slot $h(K)$, locate the record containing key K using a collision resolution policy

Hash Functions

- Which hash function is good?
  - The one that gives equal probability to all the slots, regardless of the incoming data
- Let M=16, and h(K) = K % 16. Is h(K) good?
  - No!
  - Problem 1) If the input keys are even numbers, then only 50% of the hash table space is used.
  - Problem 2) It uses only the last 4 bits of the key. If we have numbers whose last 4 bits are all the same, then they will map to a same position.

Hash Functions

- Solving Problem 1) If the input keys are even numbers, then only 50% of the hash table space is used.
  - Use a prime number for M; it helps all the hash table spaces be used.
  - Why?
    - Assume that all the keys are mapped to positions of c * i (i = 0, 1, 2, …, M-1) for a constant c
    - If M and c are relatively prime (which is true in most cases if M is prime), then c * i (i = 0, 1, 2, …, M-1) will be mapped to M different slots!
    - (Proof by contradiction)
      - Assume $c*i \equiv c * j \pmod{M}$ where $i \neq j$ in [0, M-1]. Since c is relatively prime to M, this implies i-j is a multiple of M; but it cannot happen since $i \neq j$ in [0, M-1], thus contradiction.

Hash Functions

- Solving Problem 2) It uses only the last 4 bits of the key: h(K) may be skewed for some data
- Solution: use all the bits of the key
  - 1) Mid-square method for a numerical key: square the key, take the middle r bits, and perform % M

```
    4567
    4567
-------
   31969
  27402
 22835
18268
-------
20857489
```

Hash Functions

- Solving Problem 2) It uses only the last 4 bits of the key: h(K) may be skewed for some data
- Solution: use all the bits of the key
  - 2) folding approach: sum up all the bytes, and perform %M

```
int h(String x, int M) {
  char ch[];
  ch = x.toCharArray();
  int xlength = x.length();

  int i, sum;
  for (sum=0, i=0; i<x.length(); i++)
    sum += ch[i];
  return sum % M;
}
```

Limitation: if M is large, then h() may be skewed

Hash Functions

- Solving Problem 2) It uses only the last 4 bits of the key: h(K) may be skewed for some data
- Solution: use all the bits of the key
  - 3) improved folding approach: sum up 4-byte numbers, perform %M **(Assume s.length() is a multiple of 4)**

```
long sfold(String s, int M) {

  int intLength = s.length() / 4;
  long sum = 0;
  for (int j = 0; j < intLength; j++) {
    char c[] = s.substring(j*4,(j*4)+4).toCharArray();
    long mult = 1;
    for (int k = 0; k < c.length; k++) {
      sum += c[k] * mult;
      mult *= 256;
    }
  }
  return(Math.abs(sum) % M);
}
```

What you need to know

- Concept and heuristics of self organizing list which speeds up searching
  - Count, move to front, transpose
- Motivation and main idea of hashing
  - Search a key in a constant time
- Choosing good hash functions
  - Use prime number for modulo operation
  - Use all the bits of keys

Questions?